AX-DBN: An Approximate Computing Framework for the Design of Low-Power Discriminative Deep Belief Networks

Ian Colbert, Ken Kreutz-Delgado and Srinjoy Das[1]

***Abstract*— The power budget for embedded hardware implementations of Deep Learning algorithms can be extremely tight. To address implementation challenges in such domains, new design paradigms, like Approximate Computing, have drawn significant attention. Approximate Computing exploits the innate error-resilience of Deep Learning algorithms, a property that makes them amenable for deployment on low-power computing platforms. This paper describes an Approximate Computing design methodology, AX-DBN, for an architecture belonging to the class of stochastic Deep Learning algorithms known as Deep Belief Networks (DBNs). Specifically, we consider procedures for efficiently implementing the Discriminative Deep Belief Network (DDBN), a stochastic neural network which is used for classification tasks, extending Approximation Computing from the analysis of deterministic to stochastic neural networks. For the purpose of optimizing the DDBN for hardware implementations, we explore the use of: (a) Limited precision of neurons and functional approximations of activation functions; (b) Criticality analysis to identify the nodes in the network which can operate at reduced precision while allowing the network to maintain target accuracy levels; and (c) A greedy search methodology with incremental retraining to determine the optimal reduction in precision for all neurons to maximize power savings. Using the AX-DBN methodology proposed in this paper, we present experimental results across several network architectures that show significant power savings under a user-specified accuracy loss constraint with respect to ideal full precision implementations.**

I. Introduction

In recent years, there has been a significant increase in the use of Deep Learning algorithms for a variety of cognitive computing applications such as image recognition, text retrieval and pattern completion [1]–[3]. Enabling such algorithms to operate on low-power, real-time platforms such as mobile phones and Internet of Things (IoT) devices is an area of critical interest. On such platforms, a training procedure is typically performed on a cloud server. This training involves optimizing the parameters of a cloud-based neural network using data presented to the device and uploaded to the cloud. Thereafter, the cloud performs classification when requested by the device, and subsequently the device communicates with the cloud each time an inference task is requested. This purely cloud-based approach to performing inference has a number of drawbacks including high power consumption, latency, security, and reliance on stable and fast Internet connectivity, and is therefore not suitable for use on ultra low-power devices with battery life constraints. Implementing inference locally on embedded hardware is perceived to be a desirable solution to this problem and is the focus of current research [4]–[7].

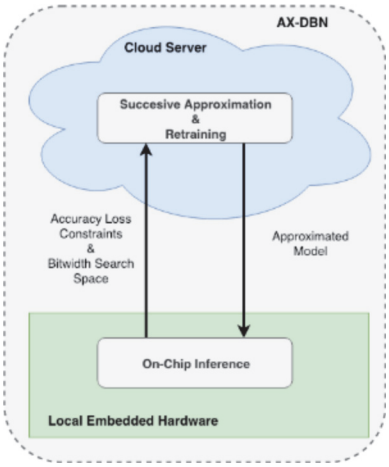

Fig. 1: **Information Flow of the AX-DBN Methodology.** The macro-level design of our framework leaves all training, approximation, and retraining to the cloud server. The hardware client sends a request to the cloud server giving it an accuracy loss constraint and a fixed bitwidth search space. The resulting approximated model is then used for energy efficient inference on embedded hardware.

We propose a shared approach where the cloud performs neural network training while a low-power implementation of the neural network on a local device is used for performing inference, as shown in Fig. 1. Neural networks are inherently error-resilient [8] and, therefore, high precision of arithmetic representations and operations is not necessary to generate sufficiently accurate performance of such algorithms. This property is exploited by Approximate Computing techniques based on the use of limited precision representations and algorithm-level approximations to achieve energy efficient implementations with negligible performance loss [4], [5]. In this paper, we propose AX-DBN, an Approximate Computing methodology for a class of stochastic neural networks known as Discriminative Deep Belief Networks (DDBNs). These are multi-layered extensions of the Discriminative Restricted Boltzmann Machine (DRBM) [3], which can be used for classification in both supervised and semi-supervised settings. Our proposed AX-DBN framework extends the analysis of deterministic networks [4], [5] to the domain of stochastic neural networks. AX-DBN involves training and approximating on the cloud and performing classification on embedded hardware, as depicted in Fig. 1. In this approach, efficiency arises from exploiting arithmetical and functional approximations subject to a user-specified accuracy loss

*Partially supported by NSF & UCSD Calit2 Pattern Recognition Lab.

[1]Communicating Author; email: s2das@ucsd.edu. All authors are affiliated with UC San Diego Pattern Recognition Lab.

constraint relative to an ideal full precision implementation.

Related Prior Work. Previous work for neural network implementations in hardware using Approximate Computing have focused on feedforward deterministic networks solving classification tasks [4], [5]. Venkataramani et al. [4] propose a design-space exploration framework, AxNN, that systematically applies approximation techniques to deterministic neural networks. Using "resilience characterization," their approach identifies and applies limited precision approximation on neurons whose impact will be the lowest on overall network performance to enable power optimized classification. Zhang et al. [5] follows a similar approach where the formulation of their neuron criticality analysis is applied to identify neurons where approximate multiplier circuits are instantiated to reduce power during classification in feedforward neural networks. Their Approximate Computing framework ApproxANN, along with their power model, addresses energy savings for memory accesses and computation workloads based on network structures and hardware characteristics.

Contributions of this Paper. Extending the work of [4] and [5] into the stochastic domain, we develop a methodology allowing for the design of an energy efficient implementation of a stochastic Discriminative DBN (DDBN) to balance the trade-off between performance and power.[1] Power efficiency arises from selectively exploiting arithmetical and functional approximations subject to a user-specified accuracy loss constraint. To accomplish this, two different levels of approximation are considered in the hardware implementation of a Discriminative DBN: (1) limited precision representation of hidden neurons; and (2) functional approximation of activation functions. The main contributions of our work are as follows:

- The use of criticality analysis to rank order neurons based on their contribution to DDBN network performance. This work carries criticality analysis to the domain of stochastic Deep Learning algorithms implemented on finite precision digital hardware. Criticality metric driven approximation using Cross Entropy is compared against random ordering using Monte Carlo simulations to gauge their effectiveness in limited precision network approximation with variable bitwidths for individual neurons.
- The use of a greedy retraining procedure to optimize neuron bitwidths under given accuracy loss constraints with respect to ideal full precision implementations.
- The use of a generalized power model for both computation workloads and memory accesses based on fixed point representations of individual neurons, number of samples and hardware characteristics.

Outline of the Paper. In Section II we review Discriminative RBMs (DRBMs) and describe how they are stacked to form Discriminative Deep Belief Networks (DDBNs). We also present the network structures that are implemented in this paper. Section III describes how limited precision and activation function approximation will be performed for the purpose of implementing such networks on hardware. In Section IV we present the mathematics for gradient-based neuron criticality analysis in stochastic neural networks such as DRBMs and DDBNs. In Section V we outline our AX-DBN hardware design methodology. Section VI outlines the power model used for this work. Section VII describes the accuracy and power results for different DRBM and DDBN architectures using our design methodology and resulting conclusions. This work is a significant extension and improvement from our preliminary results reported in [9].

[1] Although the stochastic Deep Belief Network can perform both classification and generation, in this paper we analyze its classification properties, leaving an analysis of its generation properties for future work.

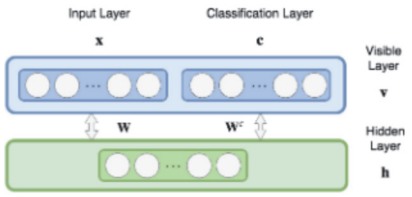

Fig. 2: **Discriminative RBM Architecture**: The visible neurons $\boldsymbol{v}$, made up of input neurons $\boldsymbol{x}$ and classification neurons $\boldsymbol{c}$, are connected to the hidden neurons $\boldsymbol{h}$ by weights $\mathbf{W}$ and $\mathbf{W}_c$, respectively.

II. DISCRIMINATIVE DEEP BELIEF NETWORKS AND NETWORK ARCHITECTURE

Here, we review the mathematical basics of the DRBM, the building block of DDBNs, and present the DRBM and DDBN configurations used for this work.

A. Discriminative Restricted Boltzmann Machines. A Restricted Boltzmann Machine (RBM) is a generative stochastic artificial neural network that can learn a probability distribution over its set of inputs. The RBM is a bipartite graph that consists of two layers - one visible layer $\boldsymbol{v}$ where the states of the units in this layer can be observed, and one hidden layer $\boldsymbol{h}$. The probability distribution implemented by the RBM is given by the Boltzmann distribution defined by:

$$P(\boldsymbol{v},\boldsymbol{h}) = \frac{e^{-E(\boldsymbol{v},\boldsymbol{h})}}{\sum\limits_{\boldsymbol{v},\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}} \tag{1}$$

where the energy function $E(\boldsymbol{v},\boldsymbol{h})$ is given below:

$$E(\boldsymbol{v},\boldsymbol{h}) = -\boldsymbol{v}^T\boldsymbol{b}^v - \boldsymbol{h}^T\boldsymbol{b} - \boldsymbol{v}^T\boldsymbol{W}\boldsymbol{h} \tag{2}$$

Here, $\boldsymbol{b}^v$ and $\boldsymbol{b}$ are biases for the visible and hidden layers, respectively, and $\boldsymbol{W}$ is the weight matrix between them [10]. From Eq. 1, one can derive the conditional distributions:

$$P(h_j|\boldsymbol{v}) = \sigma(b_j + \sum_i v_i W_{ij}) \tag{3}$$

$$P(v_i|\boldsymbol{h}) = \sigma(b_i^v + \sum_j h_j W_{ij}) \tag{4}$$

where $\sigma(x)$ is the logistic sigmoid function $1/(1+\exp(-x))$. The training method we use is described in [10], where the Contrastive Divergence (CD) method is applied to provide weight and bias updates. To use RBMs as classifiers,

Larochelle et al. [3] proposed the use of Discriminative RBMs (DRBMs), which jointly concatenate the input $\boldsymbol{x}$ and its associated "one-hot" target class $\boldsymbol{c}$ to form the single visible layer $\boldsymbol{v}$, as illustrated in Fig. 2. The energy function defined by [3] is given below:

$$E(\boldsymbol{x},\boldsymbol{h},\boldsymbol{c}) = -\boldsymbol{x}^T\boldsymbol{b}^x - \boldsymbol{h}^T\boldsymbol{b} - \boldsymbol{c}^T\boldsymbol{b}^c - \boldsymbol{x}^T\boldsymbol{W}\boldsymbol{h} - \boldsymbol{c}^T\boldsymbol{W}^c\boldsymbol{h} \quad (5)$$

Here, $\boldsymbol{b}^x$, $\boldsymbol{b}$ and $\boldsymbol{b}^c$ are biases of the visible, hidden, and classification layers, respectively. $\boldsymbol{W}$, $\boldsymbol{W}^c$ are the weight matrices between layers as shown in Fig. 2. The inputs $\boldsymbol{x}$ to the DRBM can be either binary or continuous. In the scope of this work, we consider binary inputs $\boldsymbol{x}$ as they can be used for efficient implementations on finite precision hardware with low power and area.

Similar to the RBM, the following conditional distributions can be derived for a DRBM with V visible, H hidden and C class units [3]:

$$P(h_j|\boldsymbol{v}) = P(h_j|\boldsymbol{x},\boldsymbol{c}) = \sigma(b_j + \sum_{i=1}^{V} x_i W_{ij} + \sum_{i=1}^{C} c_i W^c_{ij}) \quad (6)$$

$$P(x_i|\boldsymbol{h}) = \sigma(b^x_i + \sum_{j=1}^{H} h_j W_{ij}) \quad (7)$$

$$P(c_i|\boldsymbol{h}) = \text{softmax}(b^c_i + \sum_{j=1}^{H} h_j W^c_{ij}) \quad (8)$$

where $\sigma(x)$ is the logistic sigmoid function $1/(1+\exp(-x))$ and $\text{softmax}(x_i) = \exp(x_i)/\sum_{j=1}^{k}\exp(x_j)$ where $i \in \{1,\cdots,k\}$. Similar to the RBM, the DRBM can also be trained using Contrastive Divergence to estimate the network weight and bias values [3]. Using the notation developed by Larochelle et al. [3], the DRBM conditional probability of a class label c_i given input $\boldsymbol{x}$ is

$$P(c_i|\boldsymbol{x}) = \frac{e^{-F(\boldsymbol{x},c_i)}}{\sum_{j=1}^{C} e^{-F(\boldsymbol{x},c_j)}} \quad (9)$$

where $F(\boldsymbol{x},c_i)$ denotes the Free Energy of the DRBM given input $\boldsymbol{x}$ and class label c_i, as defined below:

$$F(\boldsymbol{x},c_i) = -b^c_{c_i} - \sum_{j=1}^{H} \log\left(1+\exp(b_j + W^c_{jc_i} + \sum_{i=1}^{V} W_{ji}x_i)\right) \quad (10)$$

A trained DRBM can be used to perform classification using two equivalent methods [11]:

- *Free Energy:* From Eq. 9, it can be seen that for a given visible vector $\boldsymbol{x}$ the class $\boldsymbol{c}$ that has the highest probability of activation corresponds to the minimum value of $F(\boldsymbol{x},c_i)$. Therefore, in this method, the Free Energy for a given $\boldsymbol{x}$ is calculated for all labels c_i. The classification result is the class c_i that corresponds to the minimum Free Energy.
- *Gibbs Sampling:* The binary activation state of each class can also be found by repeatedly sampling all class neurons of the DRBM for a given visible vector. The class with the highest activation frequency given a sufficient number of sampling iterations is the classification result.

The Gibbs Sampling method, with suitable approximations for the sigmoid function, is more amenable to realization on digital hardware than Free Energy due to the highly nonlinear dependence of Free Energy on its input values. Therefore in this work, we use Gibbs Sampling for inference in our hardware implementations of DRBMs. Details of the sigmoid function approximation are outlined in Section III.

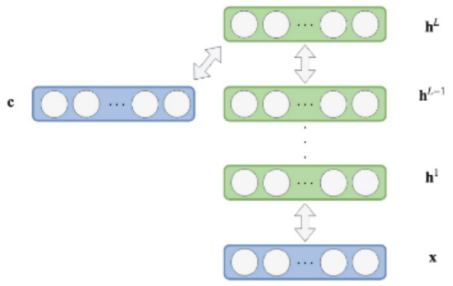

Fig. 3: **Discriminative DBN Architecture.** The DDBN architecture consists of stacked RBMs with a DRBM at the classification layer. Similar to the DRBM, the visible neurons $\boldsymbol{v}$ are split into input neurons $\boldsymbol{x}$ and class neurons $\boldsymbol{c}$. Each hidden layer $\boldsymbol{h}^l$, where $l \in \{1,\cdots,L\}$, is greedily trained layer by layer [11].

B. Discriminative Deep Belief Networks. Deep Belief Networks (DBNs) are probabilistic generative models which learn to extract a deep hierarchical representation from the training data. Hinton et al. [11] shows that DBNs are stacked RBMs and can be learned in a greedy manner by sequentially learning RBMs. Using Contrastive Divergence, the first layer is trained as an RBM with the input of the DBN as its input layer and, after the first RBM is trained, the weights $\boldsymbol{W}^1$ are fixed, as well as representations of $\boldsymbol{h}^1$. The binary states of the first hidden unit layer are then used as inputs training the second RBM for $\boldsymbol{W}^2$. Iterating this way layer-by-layer, the DBN can be trained by greedily training RBMs.

Similar to the DRBM, a DBN can also be used as a discriminative model by concatenating a classification layer to the final hidden layer, as shown in Fig. 3. Discriminative DBNs (DDBNs) can be used to perform classification using either Free Energy or Gibbs Sampling, as described for DRBMs. With the Free Energy method, we compute the activation probability of all hidden units across layers. Following this, the classification result is given by the class $\boldsymbol{c}$ that has the minimum Free Energy across all classes based on the activation values from the last hidden layer L. Using Gibbs Sampling, the binary activation state of each class given a visible vector $\boldsymbol{x}$ is found by repeated sampling of all hidden neurons across all layers and class neurons. The correct classification result is given by the class with the highest activation frequency. Similar to the DRBM, our hardware implementations of DDBNs use Gibbs Sampling for on-chip classification.

C. Network Architecture. We use the Discriminative DBN to perform classification on the MNIST dataset, which contains 60k training samples and 10k test samples of 28x28 gray-scale handwritten digits. We binarize the images using a fixed threshold of 0.5. Throughout this paper, our DRBM and DDBN naming conventions denote the amount of neurons in

each hidden layer bottom up, e.g. DDBN-100-200 denotes a 2 layer DDBN with 100 neurons in the first hidden layer and 200 neurons in the second hidden layer. Our analysis in this paper is based on the following architectures:

- 300 neuron budget: DRBM-300, DDBN-100-200
- 600 neuron budget: DRBM-600, DDBN-100-200-300

III. Limited Precision & Function Approximation

Discriminative DBNs implemented with full precision weights and biases and nonlinear sigmoidal activation functions cannot be directly implemented for inference on finite precision digital hardware platforms. We therefore study the effect of fixed-point representations and approximate sigmoid functions on DDBN classification performance. These approximations form the basis of our AX-DBN framework used for the implementation of DDBNs in finite precision embedded hardware.

A. Limited Precision Approximation. Fixed-point is preferred to floating-point representation because the latter requires extensive area and power for digital hardware implementations [12]. As shown in Fig. 4, there are three steps in the computation where fixed-point variables can be used to map the network onto its limited precision (LP) version:

- LP1 limits the precision of weight $\boldsymbol{W}$ and bias $\boldsymbol{b}$
- LP2 limits the precision of the quantity $\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$
- LP3 limits the precision of $\sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$

B. Sigmoid Function Approximation. Approximate implementations and comparisons of sigmoid activation functions designed for digital hardware exist in the literature [13], [14]. Here we use PLAN, a piecewise linear approximation of the sigmoid function proposed by Amin et al. [14] in our design

$$y = \begin{cases} f(x) & x > 0 \\ 1 - f(|x|) & x \leq 0 \end{cases} \tag{11}$$

where

$$f(x) = \begin{cases} 1 & x > 5 \\ 0.03125x + 0.84375 & 2.375 < x \leq 5 \\ 0.125x + 0.625 & 1 < x \leq 2.375 \\ 0.25x + 0.5 & 0 \leq x \leq 1 \end{cases}$$

Only addition and shift operations are involved in the approximation given by Eq. 11 which allows for a relatively inexpensive implementation in digital hardware with minimal accuracy loss.

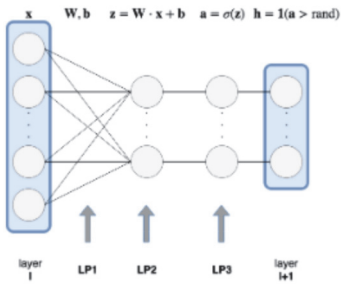

Fig. 4: **Stages of Limited Precision Approximation.** We can exploit limited precision (LP) approximations at various stages in the network.

IV. Neuron Ordering using Criticality Analysis

A neuron is said to be *critical* if a network's performance is significantly degraded by random noise injected on said neuron, otherwise it is said to be *resilient* [5]. The performance of a neural network is invariant to lowering the precision of, or even removing, resilient neurons, whereas it is significantly degraded otherwise. In the AxNN [4] and ApproxANN [5] design methodologies, which use deterministic feedforward networks, the Euclidean distance between the classification output and true label is used as the loss function for both training and criticality analysis. The magnitude of the average loss over all training samples is used to characterize the criticality of each neuron.

For the case of stochastic neural networks, such as DRBMs and DDBNs, in this paper we perform the following steps for criticality analysis:

- After performing stochastic learning, we estimate the criticality of individual neurons using a gradient-based backpropagation approach as outlined in [4] and [5]. However, unlike deterministic models, DRBMs and DDBNs are fundamentally stochastic models which can in principle be used for both discrimination and generation tasks. Keeping this perspective of a fully probabilistic framework, we propose the use of Cross Entropy as a loss function for determining critical neurons using gradient-based backpropagation.
- Inference in DRBMs and DDBNs is performed by Gibbs Sampling as described in Section II for ease of implementation in digital hardware. However, for criticality analysis we represent the binary states of the neurons in these networks with their activation probabilities, which are continuous between 0 and 1.

In the Cross Entropy (CE) loss function, y_j and a_j^c denote the one-hot ground truth and softmax class prediction probabilities, respectively.

$$\mathbb{L}_{\text{CE}} = -\sum_j y_j \ln\left(a_j^c\right)$$

The derivative of a given loss function with respect to the value of a hidden neuron relates that neuron's contribution to classification error caused by bitwidth and functional approximations at that neuron. Using CE, the error sensitivity of loss due to corruption of neuron j in layer $\boldsymbol{h}^L$ (denoted as h_j^L) of a DDBN with L layers is defined in Eq. 12, where a_i^c is defined by the softmax output in Eq. 8 and $z_i^L = b_i^c + \sum_j h_j^L W_{ij}^c$.

$$\begin{aligned} \frac{\partial \mathbb{L}_{\text{CE}}}{\partial h_j^L} &= \sum_i \frac{\partial \mathbb{L}}{\partial a_i^c} \frac{\partial a_i^c}{\partial z_i^L} \frac{\partial z_i^L}{\partial h_j^L} \\ &= \sum_i (-\frac{y_i}{a_i^c}) \cdot \left(\frac{\partial \text{softmax}(z_i^c)}{\partial z_i^c} \frac{\partial \sum_k (h_k^L W_{ik}^c + b_l^c)}{\partial h_j^L} \right) \\ &= \sum_i (-\frac{y_i}{a_i^c}) \cdot a_i^c (\mathbb{1}_{i=j} - a_j^c) W_{ij}^c \end{aligned} \tag{12}$$

The error sensitivity of neuron j in hidden layers $\boldsymbol{h}^l$, where $l < L$, is computed by Eq. 13. The binary states of individual neurons h_i^l are approximated by their sigmoid output a_i^l. Here, $\sigma(z_i^l) = 1/(1+\exp(-z_i^l))$.

$$\begin{aligned}\frac{\partial \mathbb{L}_{\mathrm{CE}}}{\partial h_j^l} &= \sum_i \frac{\partial \mathbb{L}}{\partial a_i^{l+1}} \frac{\partial a_i^{l+1}}{\partial h_j^l} \\ &= \sum_i \frac{\partial \mathbb{L}}{\partial a_i^{l+1}} \cdot \sigma(z_j^{l+1})(1-\sigma(z_j^{l+1}))W_{ij}^{l+1}\end{aligned} \tag{13}$$

For k samples of training data, the criticality score for neuron j in hidden layer $\boldsymbol{h}^l$ with $N_{\boldsymbol{h}^l}$ neurons, where $l \in \{1, \cdots, L\}$, is given by:

$$\text{criticality}(h_j^l) = \frac{\mathbb{E}[h_j^l]}{\frac{1}{N_{\boldsymbol{h}^l}} \sum_{i=1}^{N_{\mathbf{h}^l}} \mathbb{E}[h_j^l]} \tag{14}$$

where $\mathbb{E}[h_j^l] = \frac{1}{k} \sum_{\mathbf{x},\mathbf{y} \in \text{data}} \left| \frac{\partial \mathbb{L}_{\mathrm{CE}}}{\partial h_j^l} \right|$.

The hidden neurons are then ranked in order from least to most critical by the magnitude of these obtained scores.

V. AX-DBN Design Methodology

The design methodology illustrated by Fig. 1 is composed of two parts: (1) a cloud-based training and approximation process that optimizes the precision of individual neurons for inference on hardware; (2) inference performed locally on embedded hardware.

A. Cloud-based Model Training and Approximation. The approximation algorithm is a scalable framework that can be applied to any fully connected network of variable width and depth. Figure 5 illustrates the approximation flow chart, starting with a user-specified accuracy loss constraint with respect to a full precision implementation and a set of allowed bitwidths for a specified DRBM/DDBN model then ending with its approximated counterpart. Algorithm 1 formally describes this approximation procedure and can be summarized by two stages:

1) Full precision training and uniform bitwidth reduction
2) Neuron bitwidth reduction using (a) neuron criticality analysis and retraining and (b) limited precision neuron approximation

Stage 1 first trains the specified model at full precision then subsequently reduces bitwidth uniformly across all weights and biases until it can no longer maintain the accuracy loss constraint. Stage 2 first analyzes and rank orders hidden neurons according to a given criticality metric, then individually approximates these neurons based on the criticality ranking and, finally retrains the limited precision model to allow for further network approximation. In our experiments, hidden neurons can be represented at 64-bit $Q8.56$, 16-bit $Q8.8$, 12-bit $Q6.6$, 8-bit $Q4.4$, or 4-bit $Q1.3$ precision.[2] Here, we use $Qm.n$ to denote the fixed-point precision format with m integral bits (including sign bit), and n fractional bits. Additionally, the algorithm can prune the neuron completely. Class neurons are represented at 16-bit $Q8.8$ precision and not considered in our optimization due to their relatively minor impact on power savings when compared to that arising from approximating the larger number of hidden neurons considered in our architectures.

[2]We found that these $Qm.n$ bitwidths gave the best accuracy when approximating the models uniformly.

Algorithm 1 AX-DBN Approximation Algorithm. The approximation algorithm greedily explores neuron bitwidth distributions given a specified model (model_0), an accuracy constraint ($\text{acc}_{\min}$), and a set bitwidth search space (bw). Algorithm hyper-parameters include the desired neuron step size (numc), the number of approximated resilient neurons (numr), the total number of hidden neurons (numhid). The summed difference in all neurons bitwidths between successive criticality-retraining iterations is Δsumbit.

Input: *specified model* [model_0], *min accuracy* [$\text{acc}_{\min}$], *bitwidth search space* [bw]

```
model ← fullprecisionTraining(model_0)
model_ax ← uniformApprox(model)
while Δsumbit > 0 do
    order ← criticalityScore(model_ax)
    numr ← numhid
    while numr > 0 and acc > acc_min do
        model_ax ← neuronApprox(model_ax, numr, order)
        numr ← numr − numc
    end while
    model_ax ← retrain(model_ax)
end while
```

Output: *approximated model* [model_{ax}]

B. Inference in Hardware. After the cloud-based approximation procedure is completed, we download the limited precision weights and biases onto embedded hardware and perform Gibbs Sampling for local inference, as described in Section II. As depicted in Fig. 6, our hardware implementations use a pipelined architecture consisting of $L+1$ stages with one stage for each of the L hidden layers and a classification layer, respectively.

VI. Power Model for Computation Workload and Memory Access

To perform inference on a DDBN with x visible units, c class units, k data samples, and L hidden layers each with N_{h_l} hidden neurons, a chip needs to read network parameters and input data from memory and to write classification results to memory. Each weight or bias is represented by an q-bit fixed-point number, each input sample is a x-dim binary vector, and each classification result is represented by a c-dim binary vector. In this model, q refers to $m+n$ in a $Qm.n$ representation and the number of neurons at q-bit precision in hidden layer $\boldsymbol{h}^l$ is given by $N_{h_l}^q$.

The power for off-chip to on-chip data transfer DT is given by Eq. 15, where the power consumption for reading and writing 1 bit of data is given by A. The computation workload CW is defined by Eq. 16 in terms of multiply-accumulate operations, where power consumption for a q-bit ALU operation is given by B_q. A is obtained by modeling in CACTI [15]. We calculate B_q as the average accumulator

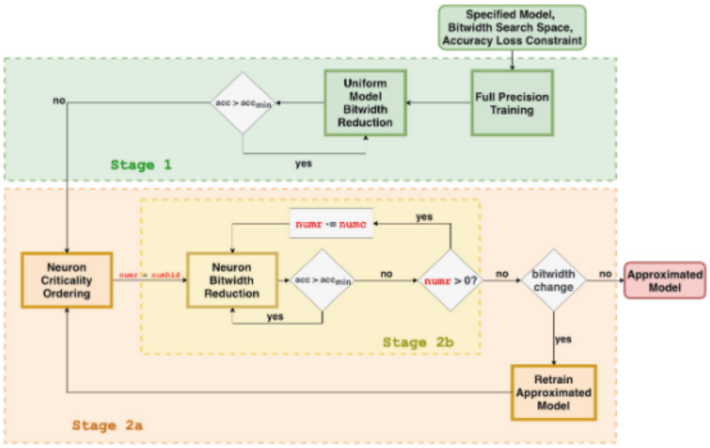

Fig. 5: **Cloud Server Network Approximation Design Flow.** The approximation algorithm can be broken into two stages: (1) uniform bidwidth reduction and (2) neuron criticality analysis with systematic bitwidth reduction and retraining. The algorithm takes in a specified model, a set bitwidth search space, and an accuracy loss constraint as inputs then returns an approximated energy-efficient model. All algorithm hyperparameters are shown in red. The number of hidden neurons is given by **numhid**, the number of resilient neurons to approximate is given by **numr**, and the search step size if given by **numc**.

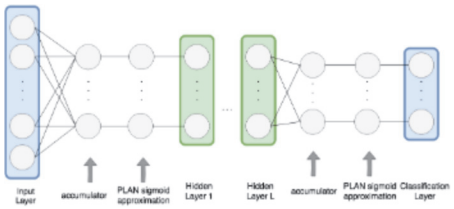

Fig. 6: **DDBN Inference on Local Embedded Hardware.** In our hardware implementations, we use Gibbs Sampling to propagate binary activation states from the input layer $\boldsymbol{x}$ to the classification layer $\boldsymbol{c}$ and use the PLAN approximation for each sigmoid output. Each hidden layer performs random sampling using a pseudo-random pattern generator (PRPG) implemented in hardware.

power consumption using Verilog implementations of our specified DRBM and DDBN architectures at q-bit precision with Synopsys EDA tools using a 65 nm standard cell library. We estimate the total power consumed by an AX-DBN approximated model to be the combination of its data transfer and computation workload, i.e $DT + CW$. In our implementation, q takes the following values: $\{4, 8, 12, 16, 64\}$.

VII. Experimental Results

An effective criticality metric should be able to accurately determine hidden neurons that can be approximated using lower precision representations while maintaining specified classification accuracy above a desired threshold. Our simulations show that the criticality measure based on Cross Entropy used in this paper achieves this objective. Owing to the stochasticity of our models, we run 200 Monte Carlo iterations to obtain distributions of power savings and average neuron bitwidth. A different random seed is used in each Monte Carlo iteration to initialize the weights and biases of a new model and train it using Contrastive Divergence (CD) on the MNIST dataset. We then compare the performance of criticality metric driven realizations versus that of random ordering.

We explore the effectiveness of our AX-DBN framework across the DRBM and DDBN architectures presented in Section II. For an even comparison, we fix the total number of neurons as we increase the depth of the network - i.e. DDBN-100-200 and DRBM-300 will both have a fixed budget of 300 neurons. In our experiments, we first train each network using Contrastive Divergence on the MNIST dataset, we then perform our criticality-driven approximations on each network, and finally evaluate the approximated model's performance. Reducing the bitwidth of all neurons uniformly results in significant accuracy degradation, as shown in Table I. With each median and mean, we provide the Inter-Quartile Range (IQR) and standard deviation in parenthesis, respectively. Using criticality-driven neuron approximation and pruning, we are able to significantly reduce the average neuron bitwidth while maintaining user-specified accuracy loss constraints with respect to ideal full precision implementations, as shown in Tables II and III.

Figures 7, 8, 9 and 10 visualize the power savings with respect to a 64-bit implementation, the distribution of neuron bitwidths, and average neuron bitwidth for 1% and 5% accuracy loss constraints for different network architectures. We observe that approximated networks realized using Cross Entropy driven critical neuron determination yield higher average power savings and lower average neuron bitwidths when compared to those realized using random neuron ordering. Overall, Cross Entropy is an effective metric for hidden neuron criticality determination across accuracy loss constraints and stochastic network architectures.

$$DT = A\left((x+1)\sum_{q=1}^{64} qN_{h_1}^q + \sum_{l=1}^{L-1}(N_{h_l}+1)\sum_{q=1}^{64}(qN_{h_{(l+1)}}^q) + 16(N_{h_L}+1)c + k(x+c)\right) \tag{15}$$

$$CW = k\left((x+1)\sum_{q=1}^{64}(B_q N_{h_1}^q) + \sum_{l=1}^{L-1}(N_{h_l}+1)\sum_{q=1}^{64} B_q N_{h_{(l+1)}}^q + B_{16}(N_{h_L}+1)c\right) \tag{16}$$

Architecture	4-bit	8-bit	12-bit	16-bit	64-bit
DRBM-300	54.5 (5.3)	88.6 (1.1)	94.0 (0.3)	94.3 (0.2)	94.3 (0.3)
DDBN-100-200	73.7 (4.7)	94.3 (0.5)	95.9 (0.2)	95.9 (0.2)	96.0 (0.2)
DRBM-600	54.2 (4.2)	82.7 (3.0)	95.0 (0.2)	95.3 (0.2)	95.3 (0.2)
DDBN-100-200-300	58.8 (7.4)	93.0 (0.8)	95.6 (0.2)	95.6 (0.2)	95.9 (0.2)
DRBM-300	54.6 (3.8)	88.5 (0.8)	94.0 (0.2)	94.3 (0.2)	94.3 (0.2)
DDBN-100-200	74.0 (3.8)	94.3 (0.4)	95.9 (0.1)	95.9 (0.2)	95.9 (0.1)
DRBM-600	54.0 (3.0)	82.5 (2.3)	95.0 (0.2)	95.3 (0.2)	95.3 (0.2)
DDBN-100-200-300	58.7 (5.3)	92.8 (0.7)	95.6 (0.2)	95.6 (0.2)	95.9 (0.2)

TABLE I: **Classification Accuracy at Uniform Model Bitwidths.** We run 200 Monte Carlo iterations to measure the median (top) and mean (bottom) test accuracy of each architecture when reducing the bitwidth of each neuron uniformly. With each median and mean, we provide the IQR and standard deviation in parenthesis, respectively.

Architecture	FP	Random	CE
DRBM-300	94.3 (0.3)	93.7 (0.7) | 11.1-bit (0.4)	93.6 (0.5) | 10.2-bit (0.6)
DDBN-100-200	96.0 (0.2)	95.0 (0.2) | 9.63-bit (0.9)	94.8 (0.2) | 8.72-bit (0.9)
DRBM-600	95.3 (0.2)	94.7 (0.8) | 11.3-bit (0.3)	94.5 (0.6) | 9.68-bit (0.7)
DDBN-100-200-300	95.9 (0.2)	94.8 (0.4) | 10.3-bit (0.6)	94.8 (0.3) | 8.72-bit (0.7)
DRBM-300	94.3 (0.2)	93.8 (0.5) | 11.1-bit (0.3)	93.7 (0.4) | 10.2-bit (0.4)
DDBN-100-200	95.9 (0.1)	95.0 (0.2) | 9.57-bit (0.7)	94.8 (0.2) | 8.71-bit (0.7)
DRBM-600	95.3 (0.2)	94.9 (0.5) | 11.2-bit (0.3)	94.6 (0.4) | 9.66-bit (0.5)
DDBN-100-200-300	95.9 (0.2)	94.8 (0.2) | 10.3-bit (0.5)	94.8 (0.2) | 8.71-bit (0.5)

TABLE II: **Classification Accuracy after AX-DBN Criticality-Driven Network Approximation with a 1% Accuracy Loss Constraint.** We run 200 Monte Carlo iterations to measure the median (top) and mean (bottom) test accuracy and average neuron bitwidth of each architecture when approximating with AX-DBN given a 1% accuracy loss constraint. With each median and mean, we provide the IQR and standard deviation in parenthesis, respectively. The classification accuracy of the full precision (FP) model on the test dataset is given by the first column. FP denotes the full precision 64-bit floating point model.

Architecture	FP	Random	CE
DRBM-300	94.3 (0.3)	91.8 (1.5) | 8.00-bit (0.6)	92.4 (1.0) | 6.46-bit (1.0)
DDBN-100-200	96.0 (0.2)	92.4 (1.0) | 6.91-bit (0.4)	92.5 (0.8) | 5.92-bit (0.5)
DRBM-600	95.3 (0.2)	93.5 (1.1) | 9.05-bit (0.6)	94.4 (0.7) | 6.86-bit (1.1)
DDBN-100-200-300	95.6 (0.2)	92.4 (0.9) | 7.36-bit (0.4)	92.3 (0.8) | 6.13-bit (0.4)
DRBM-300	94.3 (0.2)	91.7 (0.9) | 8.09-bit (0.4)	92.2 (0.8) | 6.52-bit (0.7)
DDBN-100-200	95.9 (0.1)	92.4 (0.6) | 6.89-bit (0.3)	92.4 (0.6) | 5.89-bit (0.3)
DRBM-600	95.3 (0.2)	93.7 (0.7) | 9.10-bit (0.5)	94.3 (0.6) | 6.81-bit (0.7)
DDBN-100-200-300	95.9 (0.2)	92.3 (0.7) | 7.38-bit (0.3)	92.2 (0.6) | 6.15-bit (0.3)

TABLE III: **Classification Accuracy after AX-DBN Criticality-Driven Network Approximation with a 5% Accuracy Loss Constraint.** We run 200 Monte Carlo iterations to measure the median (top) and mean (bottom) test accuracy and average neuron bitwidth of each architecture when approximating with AX-DBN given a 5% accuracy loss constraint. With each median and mean, we provide the IQR and standard deviation in parenthesis, respectively. The classification accuracy of the full precision (FP) model on the test dataset is given by the first column. FP denotes the full precision 64-bit floating point model.

VIII. CONCLUSIONS

In this paper, we propose a systematic approximation methodology for stochastic Discriminative Restricted Boltzmann Machines (DRBMs) and Discriminative Deep Belief Networks (DDBNs) to optimize power consumption subject to the constraint of maintaining user-specified classification accuracy. This work extends criticality analysis from the domain of deterministic neural networks to the realm of stochastic networks. Our procedure involves two key steps: (1) The use of criticality analysis to rank order neurons based on their contribution to network performance; (2) The use of greedy retraining to optimize neuron bitwidths under accuracy constraints. Our results show that Cross Entropy can be used as an effective metric for determining neuron criticality in stochastic network approximation and yields lower average neuron bitwidth representations as well as higher savings in power consumption when compared to random criticality ordering of neurons. Our future research will extend our methodology to optimize the power consumption of hardware implementations of generative neural networks for purposes such as image generation, denoising, and infilling.

ACKNOWLEDGEMENTS

This work was supported in part by NSF awards CNS-1730158, ACI-1540112, ACI-1541349, OAC-1826967, the University of California Office of the President, and the California Institute for Telecommunications and Information Technology's Qualcomm Institute (Calit2-QI). Thanks to CENIC for the 100Gpbs networks. We would also like to thank Professor Andrew Kahng, Jonas Chan and Chih-Yin Kan at UC San Diego for providing assistance with Verilog implementations and power measurements.

REFERENCES

[1] G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. Science, 313(5786):504–507, 2006.

[2] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In Proc. 24th int. conf. Machine learning, pages 791–798. ACM, 2007.

[3] H. Larochelle, M. Mandel, R. Pascanu, and Y. Bengio. Learning algorithms for the classification restricted boltzmann machine. Journal of Machine Learning Research, 13(Mar):643–669, 2012.

[4] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan. Axnn: energy-efficient neuromorphic systems using approximate computing. In Proceedings of the 2014 international symposium on Low power electronics and design, pages 27–32. ACM, 2014.

[5] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu. Approxann: an approximate computing framework for artificial neural network. In Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, pages 701–706. EDA Consortium, 2015.

[6] S. Das, B.U. Pedroni, P. Merolla, J. Arthur, A.S. Cassidy, B.L. Jackson, D. Modha, G. Cauwenberghs, and K. Kreutz-Delgado. Gibbs sampling with low-power spiking digital neurons. In 2015 IEEE Int. Symp. Circuits & Systems (ISCAS), pages 2704–2707. IEEE, 2015.

[7] Y.-H. Chen, T. Krishna, J.S. Emer, and V. Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE Journal of Solid-State Circuits, 52(1):127–138, 2017.

[8] C.M. Bishop. Neural networks for pattern recognition. Oxford university press, 1995.

[9] X. Xu, S. Das, and K. Kreutz-Delgado. Approxdbn: Approximate computing for discriminative deep belief networks. arXiv preprint arXiv:1704.03993, 2017.

[10] G.E. Hinton. A practical guide to training restricted boltzmann machines. Momentum, 9(1):926, 2010.

[11] G.E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. Neural computation, 18(7):1527–1554, 2006.

[12] D.L. Ly and P. Chow. A high-performance fpga architecture for restricted boltzmann machines. In Proc. ACM/SIGDA int. symp. Field programmable gate arrays, pages 73–82. ACM, 2009.

[13] M.T. Tommiska. Efficient digital implementation of the sigmoid function for reprogrammable logic. In Computers and Digital Techniques, IEE Proceedings-, volume 150, pages 403–411. IET, 2003.

[14] H. Amin, K.M. Curtis, and B.R. Hayes-Gill. Piecewise linear approximation applied to nonlinear function of a neural network. In Circuits, Devices & Syst., IEE Proc., volume 144, pages 313–317. IET, 1997.

[15] H. Labs. http://www.hpl.hp.com/research/cacti/.

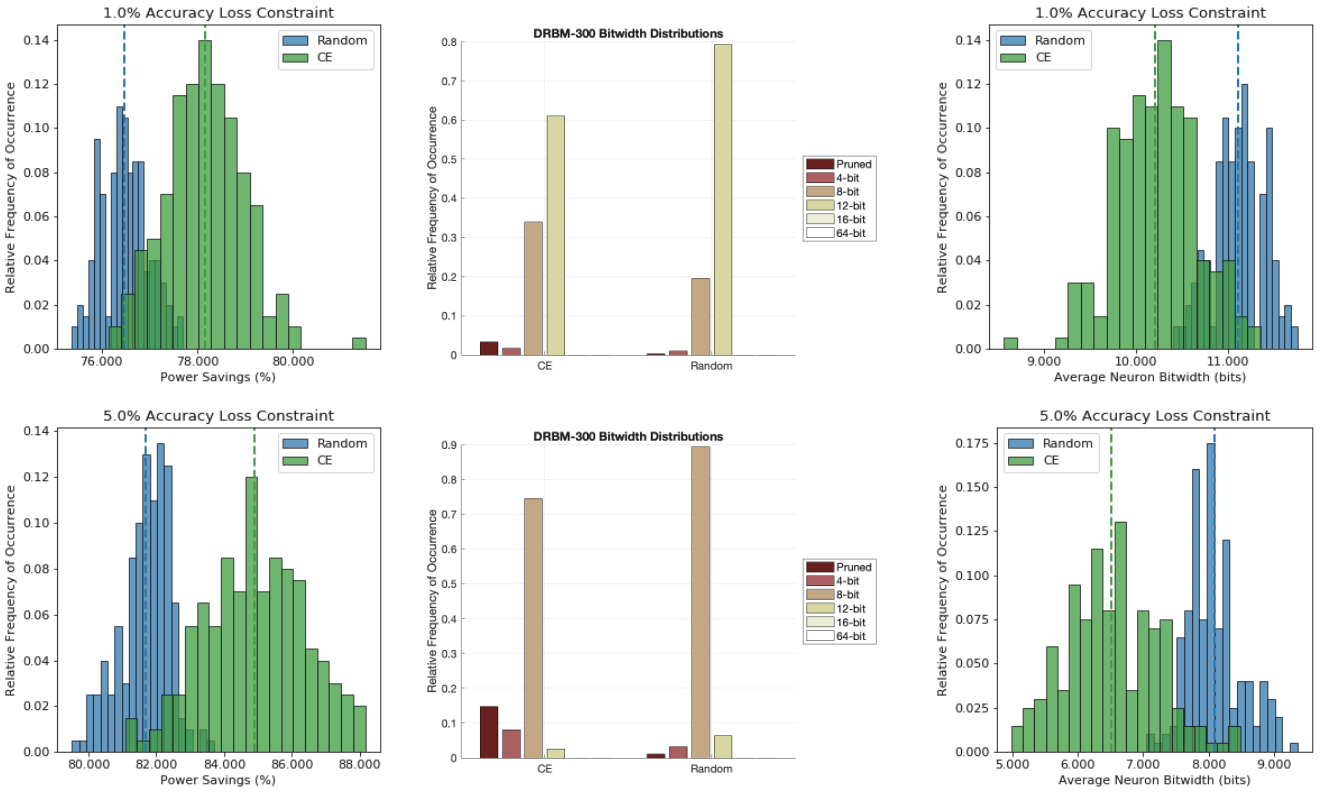

Fig. 7: **DRBM-300.** Relative power savings (left), neuron bitwidth distribution (middle) and average neuron bitwidths (right).

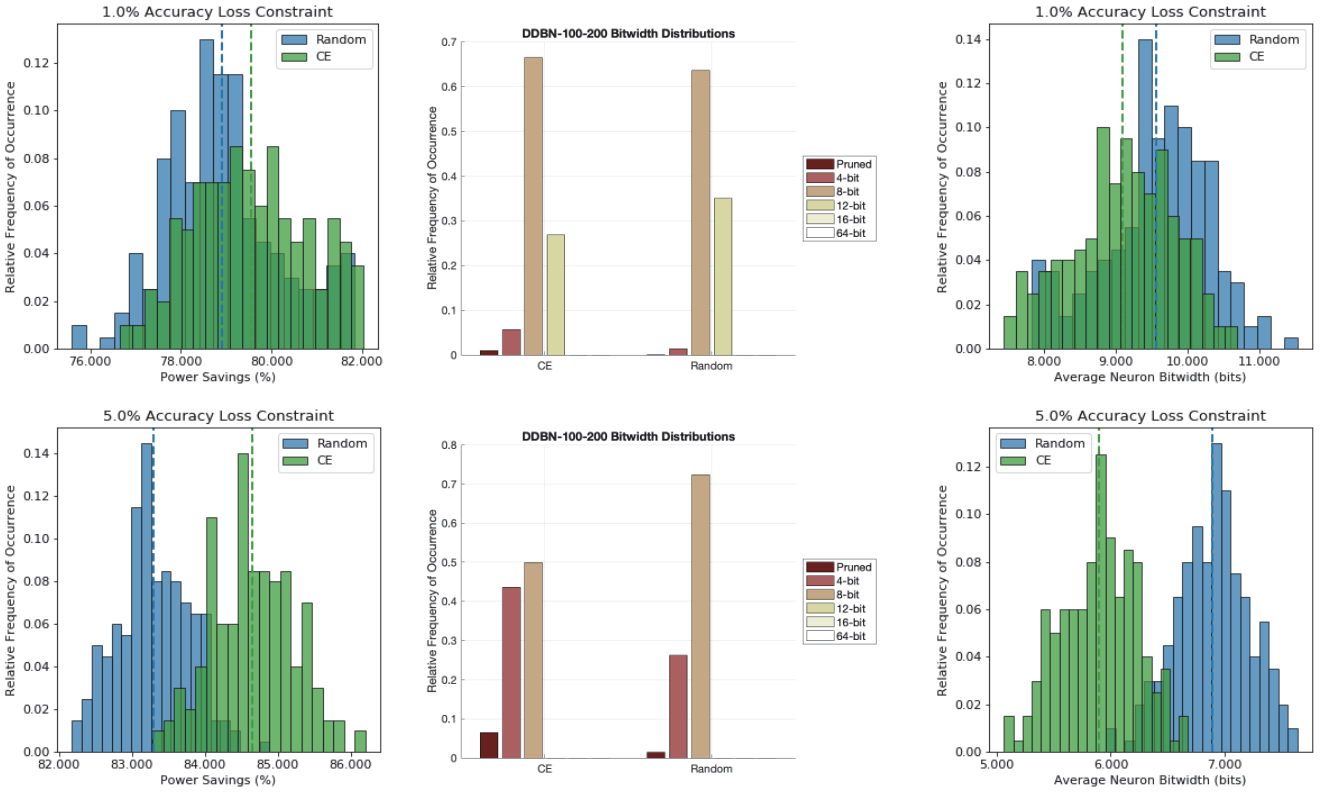

Fig. 8: **DDBN-100-200.** Relative power savings (left), neuron bitwidth distribution (middle) and average neuron bitwidths (right).

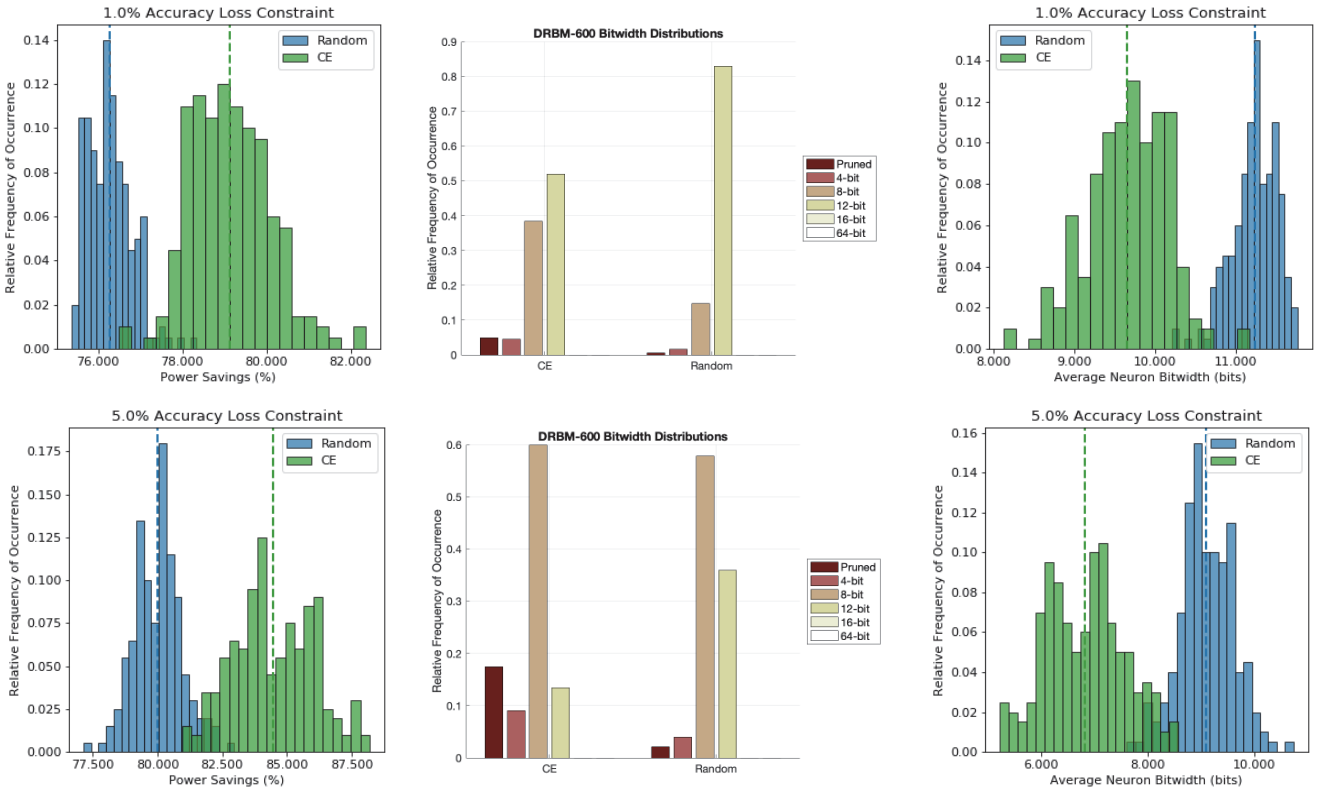

Fig. 9: **DRBM-600.** Relative power savings (left), neuron bitwidth distribution (middle) and average neuron bitwidths (right).

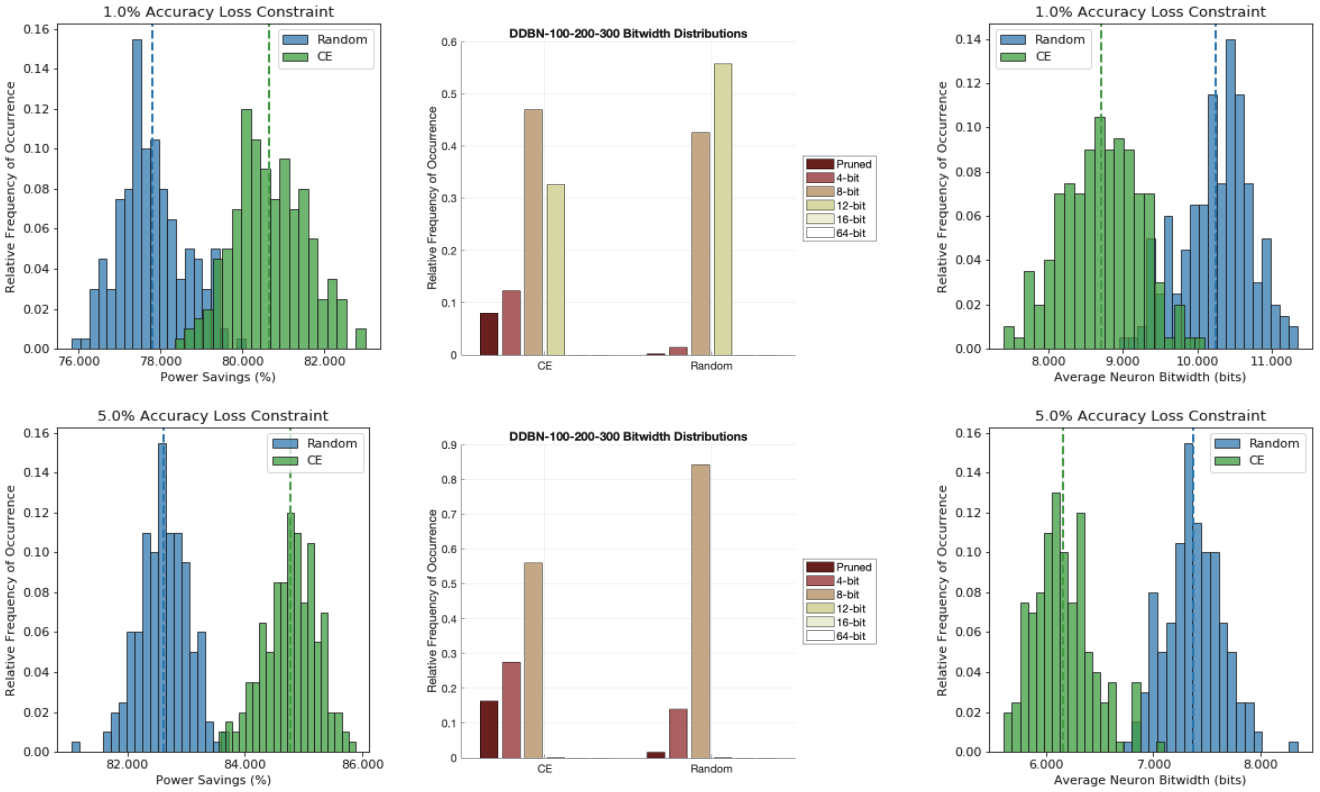

Fig. 10: **DDBN-100-200-300.** Relative power savings (left), neuron bitwidth distribution (middle) and average neuron bitwidths (right).